Rajalakshmi Engineering College

Name: HARI PRASATH PALANIMURUGAN
Email: 241801080@rajalakshmi.edu.in
Roll no: 241801080
Phone: 8072956499
Branch: REC
Department: l AI & DS FB
Batch: 2028
Degree: B.E - AI & DS

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 5_COD_Question 1

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

John is learning about Binary Search Trees (BST) in his computer science class. He wants to create a program that allows users to delete a node with a given value from a BST and print the remaining nodes using an in-order traversal.

Implement a function to help him delete a node with a given value from a BST.

Input Format

The first line of input consists of an integer N, representing the number of nodes in the BST.

The second line consists of N space-separated integers, representing the values of the BST nodes.

The third line consists of an integer V, which is the value to delete from the BST.

Output Format

The output prints the space-separated values in the BST in an in-order traversal, after the deletion of the specified value.

If the specified value is not available in the tree, print the given input values in-order traversal.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
10 5 15 2 7
15
Output: 2 5 7 10

Answer

```c
#include <stdio.h>
#include <stdlib.h>

struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};

struct TreeNode* createNode(int key) {
    struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    newNode->data = key;
    newNode->left = newNode->right = NULL;
    return newNode;
}

struct TreeNode* insert(struct TreeNode* root, int key) {
    if (root == NULL) {
        struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct TreeNode));
```

```c
        newNode->data = key;
        newNode->left = newNode->right = NULL;
        return newNode;
    }

    if (key < root->data) {
        root->left = insert(root->left, key);
    } else {
        root->right = insert(root->right, key);
    }

    return root;
}

// Function to find the minimum value node in a given BST
struct TreeNode* findMin(struct TreeNode* root) {
    while (root && root->left != NULL) {
        root = root->left;
    }
    return root;
}

// Function to delete a node with a given value from the BST
struct TreeNode* deleteNode(struct TreeNode* root, int key) {
    if (root == NULL) {
        return root;
    }

    if (key < root->data) {
        root->left = deleteNode(root->left, key);
    } else if (key > root->data) {
        root->right = deleteNode(root->right, key);
    } else {
        // Node to be deleted found

        // Case 1: Node has no children (leaf node)
        if (root->left == NULL && root->right == NULL) {
            free(root);
            return NULL;
        }

        // Case 2: Node has only one child
```

```c
        if (root->left == NULL) {
            struct TreeNode* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct TreeNode* temp = root->left;
            free(root);
            return temp;
        }

        // Case 3: Node has two children
        struct TreeNode* temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }

    return root;
}

// Function to perform an in-order traversal of the BST
void inorderTraversal(struct TreeNode* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}
int main()
{
    int N, rootValue, V;
    scanf("%d", &N);
    struct TreeNode* root = NULL;
    for (int i = 0; i < N; i++) {
        int key;
        scanf("%d", &key);
        if (i == 0) rootValue = key;
        root = insert(root, key);
    }
    scanf("%d", &V);
    root = deleteNode(root, V);
    inorderTraversal(root);
    return 0;
```

```
}
```

Status : Correct   *Marks : 10/10*